Assembler

Im Buch zum BERT „Einführung in die Mikroprozessor-Anwendung“ von Roland Schule wurde ein kleiner Assembler erwähnt, im Buch „Mikroprozessoren in der Meß- und Regeltechnik. Funktion - Aufbau und Programmierung“ von Gerhard Ledig fand sich ein Listing dieses Assemblers.

Vermutlich wurde der Assembler als „RZ8“ 1982 (?) von der Firma Arcom Control Systems Ltd. programmiert. Die Versionen des RZ8, die unter https://hackaday.io/project/169539-arcbus-z8-basic-microcontroller-boards beschrieben sind, scheinen Nachfolgeversionen zu sein; die Beschreibung RZ8-resident_assembler.htm passt und auch der Code ist analog aufgebaut.

Der Assembler RZ8 ist eine Ergänzung des Z8671-Basic/Debug und setzt dieses voraus.

Beschreibung

Das Assemblerprogramm wird in Basic-Zeilen erfasst.

INPUT FORMAT

Input comprises and ordered sequence of lines, optionally starting with a line containing a $ABS statement and ending with a mandatory line containing an END statement. Upper and lower case letters are allowed. The standard format is as follows:

```
line no (space) mnemonic (space) 1st operand (comma) 2nd operand
```

For example

```
101 ld 20,@30
```

A number will be interpreted as hex if it is preceded by a percent (%) sign. A preceding '#' (the 'sharp' or 'hash' character) indicates an immediate operand, as opposed to a register. For example

```
101 and r2,#%f0
```

Comments are not allowed.

Extra spaces or commas can be included and are generally interchangeable. Tabs can be used but will upset the output formatting. As with BASIC, where no ambiguity results, spaces can be omitted in order to conserve memory space, at the expense of readability. For example

```
103and10#%f
```

INSTRUCTIONS AND MNEMONICS

The assembler uses standard mnemonics as described in the Zilog Z8 PLZ/ASM assembly language manual, with some restrictions and exceptions as described below.

Program Origin

In the absence of a $ABS statement, code will be assembled to run in memoery starting after the BASIC text (i.e. at the location given by register pair 4). The statement

```
$ABS number
```

will force subsequent code to be assembled to run at the specified location. For example

```
100 $ABS %4800
```

For details of code storage see section OPERATION.

DEFB and DEFS

DEFB & followed by one or more numeric values will insert those values into the next consecutive memory locations of the assembled program. For example

```
DEFB & 0,%d,%a,%ff
```

DEFB „ followed by ASCII characters will insert their hex equivalent into the program For example

```
DEFB "END of program
```

DEFS followed by a numeric value will leave the given number of locations of free space in the code (i.e. increment the program location counter). Note that these locations will not be set to any value.

Labeled Jumps

In addition to accepting address values, the JR, JP, CALL instructions can be made to reference an assembler program line number by inserting the line number preceded by '*'. for example

```
JR nov,*103
```

The assembler will insert the appropriate value during assembly. For example SAMPLE

```
100 $abs %4800
101 call *110
102 ret
110 or r2,r4
111 or r3,r5
112 ret
200 end
```

OPERATION

The assembler has the entry location: %0800 On entry the register pair 20/21 should contain the hex equivalent of the line number of the first statement to be assembled, and register pair 18/19 an optional hex „offset“ which represents the difference between the location where the code is to be stored and the location at which it is assembled to run. The offset should be a multiple of 256 (%100). These registers pairs correspond to the BASIC conventions for calling machine-code subroutines. The

assembler is therefore most easily called by:

```
go@1800,line number
```

if code is to be stored at its run address, or

```
go@%1800,line number, offset
```

if code is to be stored at a different address from the run address. For example

```
 1 go@%0800,10
 2 stop
10 $abs %4400
30 ld r4,#123
40 add 10,r4
50 ret
60 end
```

when RUN, will store the code into RAM address %4400. Alternatively

```
 1 go@%0800,10,%3000
 2 stop
10 $abs %1400
30 ld r4,#123
40 add 10,r4
50 ret
60 end
```

will assemble the code to run at address %1400 but store the code into RAM at %4400 (i.e. %1400 +%3000) This code could be subsequently be blown into EPROM address at %1000 onwards. On return for the assembler, register 18/19 contain -1 if there have been no errors or, if a line is incorrect, the line number of the line containing the error. If the assembler is called by the statement.

```
LIST USR(%0800,line number,offset)
```

Where line number is that of the first line to be assembled), and line in error will be listed on the terminal and assembly will be aborted at that point. The relevant line can be corrected and the assembler rerun. Note that in a forward jump (or Call), and errors in the lines up-to the one referenced will be reported as an error at the time that the line containing the jump (or call) is being assembled.

When assembling a program to run in RAM the offset will usually be zero. If the offset is omitted when calling the assembler a value of zero is assumed.

NOTE: The assembler checks for the presence of the second parameter by comparing R20/21 and R18/19. if the values are equal it assumes that no second parameter was given. This has the side effect that in the unlikely event that two equal parameters were specified, the offset will be ignored. For example,

```
go@usr(%4800,4096,%1000)
```

will be misinterpreted. Avoid having a first line number equal to the offset!

RUNNING THE ASSEMBLER FROM A BASIC PROGRAM

The assembler can be called from within a program and the assembled code used later in the same program. SAMPLE

```
1   go@%0800,100
2   goto 1000
100 $ABS %4100
101 OR R2,R4
102 OR R3,R5
103 RET
104 END
1000    "Enter x and Y;":""
1001    inputx:inputy
1002    "x.OR.y=%";hex(usr(%4100,x,y)):""
1003    goto 1000
```

The program will assemble the code into the locations %4100 onwards and then jump to line 1000, where the code is used to perform the 'OR' function.

Beispiel

```
1 GO@%800,100
2 GOTO 1000
100 $ABS %D100
101 OR R2,R4
102 OR R3,R5
103 RET
104 END
1000 "ENTER X AND Y;":""
1001 INPUTX:INPUTY
1002 "X.OR.Y=%";HEX(USR(%D100,X,Y)):""
1003 GOTO 1000
```